Archborn: An Open Source Tool for Automated Generation of Chip Heterogeneous Multiprocessor Architectures

Sen Ma, Hongyuan Ding, Miaoqing Huang, and David Andrews
Department of Computer Science and Computer Engineering
University of Arkansas
Fayetteville, AR 72701, USA
Email: {senma, hyding, mqhuang, dandrews}@uark.edu

Abstract—Modern platform FPGAs are sufficiently dense to allow the assembly of a complete chip heterogeneous multiprocessor systems on chip (CHMPs) within a single die. Based on CHMP, every research group that sets out to explore how an application can be accelerated on an FPGA platform must firstly integrate processors, buses, memories, and support IP components into a base architecture prior to beginning their application specific analysis. Lacking of computer architecture background, many application developers will spend significant amounts of valuable research time to create and debug the base CHMP system. In this paper, we present Archborn, an open source tool that automates the generation of modifiable CHMP architectures. Archborn can be used by application developers to create a base CHMP system that can be synthesized in seconds. The systems created by Archborn can also be modified by those who wish to create fully customized CHMP systems. We show the ease and flexibility of Archborn by automatically generating three unique systems: an NUMA architecture that is modified with an Hthreads hw/sw co-designed microkernel for multithreading, and a NUMA architecture to support the OpenCL programming model. We present analysis on resource utilizations and scalability for creating systems with up to 64 processing elements (PEs).

I. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) are continuing to grow in their use within academic laboratories. Application developers and computational science researchers routinely deploy the FPGA to enable new applications in real time and distributed systems. While FPGAs provide significant advantages, individual research groups are spending unnecessary time and effort to build infrastructure in their local laboratories. It is not uncommon for students to spend six months to gain mastery of the tools before they even begin investigating their research questions. Additional months of effort can then be expended to learn advanced computer architecture topics, followed by hand building, integrating, and testing a base system. All of these efforts represent burdens of time and effort that could be eliminated through automation and sharing. In the absence of these benefits it is not uncommon for students to become frustrated with the breadth of knowledge they must learn before they can push their research. In some cases this has caused students to switch topics to more software based research to shorten their time to graduation. This situation unfortunately is not an isolated incident and is occurring across many research groups.

In the larger picture localized development does not support good science. Common sets of benchmarks, development tools and platform infrastructures are the basis for enabling fair and unbiased comparisons. Anecdotally, conference program committees are routinely being asked to evaluate the relative merits of a new FPGA based design or application without the benefit of good solid comparisons against prior art. Achieving such comparisons would require at least a doubling of engineering and development effort to rebuild the prior artifact.

It is our belief that creating base architectures, configuring multiple tool chains, and repetitive engineering design efforts can and should be automated. Automating this type of effort will enable students and researchers to address scientific questions quicker, and with an enhanced ability to increase experimentation across a broader range of configurations.

In this paper, we present Archborn, an open source tool for custom multiprocessor architecture generation based on Xilinx TCL routines in Vivado. Firstly, Archborn contains a library of high-level APIs designed to support the creation of CHMP. It frees designers from configuration details of both processors and interconnections. A heterogeneous multiprocessor generator written in Archborn APIs can generate systems up to hundreds of processing elements (PEs). Secondly, Archborn includes several standard templates as the interface of different architectures, such as Symmetric Multiprocessor (SMP), Non-Uniform Memory Access (NUMA) and Partial Reconfigurable Accelerator (PRA). Using architecture template interfaces, user can integrate various emerging programming models and build a whole CHMP system efficiently. To present the capabilities of Archborn, we take Hthreads and HOpenCL programming models as an example by integrating them into a CHMP hardware platform generated by Archborn. Specific contributions of this paper include:

- **Archborn APIs.** Sequences of low-level TCL commands with the same functionality are wrapped into a single routine to generate PEs, buses and memory components respectively. Archborn APIs are defined as open-source vendor neutral representations that can be implemented through vendor specific low-level TCL routines. User can change the detailed definition of each API to fit in different FPGAs from various vendors, as long as based on TCL interface.

- **Archborn Templates.** To simplify the CHMP generation flows, Archborn bundled various APIs together to compose multiprocessor architecture templates such as SMP, NUMA, RING and PRA. A specific architecture is automatically built through a few basic configuration parameters, like the type of Archborn template, amount of slave groups, the number of PEs on each groups and the size of shared memory. Users can modify the existing or compose new Archborn template to meet their specific requirements.
Fig. 1. MHS Descriptions and TCL Commands.

- **Rapid Prototyping Design Flow.** Based on the built Archborn template, a user can create and connect custom platform specific IP components to multi-level buses by using the returned bus list object. Therefore, designers are able to fast prototype custom hardware platforms with supports of different programming models on FPGAs.

The remainder of this paper proceeds as follows. The next section provides an overview of background and related work. In section III, we first show how Vivado TCL primitive operations can be encapsulated into high-level Archborn APIs and Archborn template interfaces. Then, we introduce an example flow to generate a ring architecture. Section IV presents how to use Archborn templates to rapidly prototype hardware platforms supporting different programming models. The evaluation results based on Hthreads and HOpenCL are given. Finally, conclusions with a discussion on the additional research are addressed.

II. BACKGROUND AND RELATED WORK

A CHMP with more PEs can assist researchers to achieve more promising results and makes it possible for researchers to explore various problems related to high-performance computing (HPC) [1], tolerant and self-healing system [2], accelerator for embedded system [3] [4], prototyping and design space exploration [5], [6], etc. There are several work focusing on CHMP automated methods. The number of PEs in an experimental CHMP system is normally limited by two facets, resource limitation and design complexity. In this work, we focus on building CHMP with soft-core processors on modern platform FPGAs.

Xilinx Platform Studio (XPS) helps hardware designers to build, connect and configure systems with embedded processors. The MHS file serves as an input to the Platform Generator tool. Later, according to MHS file synthesis tools can generate the hardware description language (HDL) definition of the system. In an MHS file, IP definitions start with `BEGIN` and terminates with `END`. The key word `PARAMETER` is used to configure parameters of the processor. Designers can change the configurations of the processor by modifying the values of parameters. The GUI does not support the generation of CHMP with more than two processors. To create a system with more than two processors, designers have to manually duplicate the configuration descriptions in MHS files.

MHS files are utilized by several research projects to create their automated system assembly methods targeting Xilinx devices. Hthreads-Cloud [7] outlined a cloud-based tool flow to automatically create complete CHMP for platform FPGAs. In [8], the authors designed a flow for developing application-specific platforms by automatically generating CHMP for different applications.

Vivado, the latest Xilinx tool suite, provides extensive functionalities for all programmable platform FPGAs. Designing CHMP in Vivado differs from that in XPS in at least three ways. Firstly, it discontinues the support of MHS files for configurations. Secondly, TCL commands are used to create and configure all components in a CHMP design. Last but not the least, it provides the interface to import a set of TCL commands as a script file to automate the design. Compared with MHS files, Figure 1(b) presents the TCL commands to create and configure a processor instance. The command `create_bd_cell` is used to create the processor and `set_property` command is used to configure the component.

Although using the TCL shell interface in Vivado is convenient, using low-level TCL routines to create a component and configure its properties in CHMP can be a tedious and error-prone task. Besides, the whole design flow requires a detailed low-level knowledge of architecture configurations and interconnects. To offload burdens from designers, a common TCL extension library for Vivado is needed. TincrCAD [9] provided a TCL extension library to assist users in the creation of custom CAD tools.

III. ARCHBORN FRAMEWORK

The Vivado TCL shell provides a basic set of routines to create and configure IP components. TCL scripts shown in Figure 1(b) provide an alternative way to release the constrain of MHS formats. However, using low-level TCL routines will involve many details for system designers and may cause distraction. Designers may shift the attention from the pure architecture design to tedious details such as bus configurations, interconnections and even memory space allocation. Designers
Archborn APIs also support generating partial reconfiguration (PR) modules. Previously, in order to generate a PR module in a CHMP, designers have to manually define PR regions and associate its region to a specific component, which is called a PR constrain. By using Archborn PR APIs that are shown in Table I, designers can call `pr_module_gen` to generate a PR module whose PR constrains will be automatically appended to the Xilinx Design Constraint (XDC) file.

B. Archborn Templates

After defining Archborn APIs, we further design several Archborn templates to build common hardware architectures. Figure 3 shows three of them. Within a symmetric multiprocessing (SMP) architecture shown in Figure 3(a), multiple Group Bus can be connected to Host Bus and several Slave PE are attached to each Group Bus. A non-uniform memory access (NUMA) architecture is shown in Figure 3(b). Different from SMP architecture where groups contains only PEs, in NUMA architecture a group contains an extra group memory. The last architecture demonstrated in Figure 3(c) is a partial reconfigurable accelerator architecture (PRA). Each hardware accelerator is defined as a partial reconfigurable region and connected to its corresponding PE through streaming interfaces. Next we use the Ring architecture shown in Figure 4 to demonstrate how to construct an Archborn template.

1) Creation of Host Processor and Bus: Although the HOST PE is not necessary for every CHMP design, all architecture templates in this work have a Host PE for a better understanding. For each `pe_gen` (steps 1, 5), Archborn API will create a PE object following parameters indicated in Table I. The name of the PE object will be returned. For instance, in the step 1, a Host PE will be created. It is configured as a MicroBlaze with 4KB private memory and without caches (data and instruction ones) or streaming ports. The object name of Host PE will store in the variable `host`. By using `Host`, the name of Host PE can be accessed, which is presented in step 3. An interconnection bus for Host PE is created by using `bus_gen` API (step 2) and the name is saved in the variable `host_bus`. In step 3, the `bus_connect` will connect Host PE with Host Bus.

2) Creation of Slave Processors and Buses: After the Host PE and bus are created, Slave buses and PEs can be created in steps 4 and 5, respectively. Since in this example, there are four groups, each containing a Slave PE and a bus, a TCL loop is used to generate all groups. Within each loop, a list of PE objects and a list of bus objects will be returned. The variable `bus_list` contains all names of bus objects and the variable `pe_list` includes all Slave PE objects. Since Host PE has already been connected to the Host Bus, it is not included in the list of PEs. Within the nested loop in step 5, the inner loop produces PEs in each group and the outer loop connects each PE with its corresponding Group Bus.

3) Architecture Connection: All group buses and host bus will be connected appropriately in step 6 to create a ring structure. In this example, the ring structure is connected following the anticlockwise direction. In order to implement a bi-direction topology, another TCL loop can be used by swapping parameters of `bus_connect`.

```
proc pe_gen {key pe_type pm_size ic_size dc_size stream} {
  if {$pe_type == mb} {
    set pe_name mb;">key nameend"
  set_property -dict [get_bd_cells $pe_name] -type ip -vlnv xilinx.com:ip:microblaze:9.5 $pe_name
  set_property -dict [list CONFIG.C_FSL_LINKS $stream] [get_bd_cells $pe_name]
  if {$ic_size == 0} {
    set_property -dict [list CONFIG.C_USE_ICACHE 0] [get_bd_cells $pe_name]
  } else {
    set_property -dict [list CONFIG.C_USE_ICACHE 1] [get_bd_cells $pe_name]
  } 
  if {$dc_size == 0} {
    set_property -dict [list CONFIG.C_USE_DCACHE 0] [get_bd_cells $pe_name]
  } else {
    set_property -dict [list CONFIG.C_USE_DCACHE 1] [get_bd_cells $pe_name]
  } 
  # Code trimmed due to limited space
  return $pe_name
}
```

Fig. 2. Implementation of a Representative Archborn API.

will benefit from bundling TCL routines into a script file for redundant design in the future. However, such TCL scripts lack the flexibility and portability. Further, due to the limitation of Vivado TCL shell, it cannot create a custom architecture using different programming models, such as Hthreads and HOpenCL. To assist system designers in creating CHMP, Archborn extracts an abstraction layer above the Vivado TCL interfaces. By targeting various programming models, it helps system designers improve design productivity and reduce complexities to generate CHMP on modern platform FPGAs.

A. Archborn APIs

Each Archborn API is a group of TCL commands that are categorized into the same functionality. As an example shown in Figure 1(b), TCL commands `create_bd_cell` and `set_property` are paired to instantiate one processor and configure with custom parameters. By using Archborn APIs, designers will be released from setting up component configurations through many mouse clicks and from detailed TCL commands with trivial primitive parameters.

Table I lists essential Archborn APIs categorized by functionalities and wrapped into single cohesive routines. As shown in Figure 2, take `pe_gen` as an example, it is defined as a TCL procedure by using TCL keyword `proc`. Six parameters are required for this API. The API routine will first create an object name concatenating the type of object and the value of parameter `key` as a suffix. `type` stands for the type of the processor (e.g., MicroBlaze or ARM), `pm_size`, `ic_size` and `dc_size` indicate the sizes of private memory, instruction cache and data cache, respectively. The last parameter, `stream`, is the number of streaming interfaces when the processor is configured as a MicroBlaze.
interface into the Hthreads system. This allows portable HDL or general purpose processors to interface simplistically with Hthreads via several read/write registers. The VHWTI operates in coordination with a small kernel (4KB) called the Hardware Abstraction Layer (HAL). The HAL transforms Hthreads system calls into simple load/store operations that access the Hthreads cores. Using general purpose processors as replacement components for custom hardware circuits offers the flexibility and the increased design productivity but at the expense of performance.

2) OpenCL: OpenCL is a framework to design parallel applications on various computation resources (e.g., CPUs, and GPUs). The current OpenCL specification is heavily influenced by GPU programming. In [10], the author presented a hybrid hardware platform combining with OpenCL-flavor programming model called HOpenCL. In this work we further simplify the design flow by using the Archborn generation framework. Figure 5(b) demonstrates the hardware platform generated by Archborn for the OpenCL programming model. Similar to the Hthreads hardware platform, computation resources are divided into multiple groups. Inside each group, multiple software OpenCL kernels run on PEs, and hardware ones execute as dedicated hardware accelerators. Private, group and global memories form the three-level memory hierarchy in order to meet the OpenCL specification. Group scheduler dispatches group partitions of one OpenCL kernel to each hardware group of computation resource. Within each hardware computation resource, item scheduler decomposes the corresponding task group into items and arranges them to software or hardware kernels for execution. Barriers provide the synchronization insides each group.

B. Design Flow with Archborn

As shown in Figure 6, design flows to generate multi-processor architectures with Archborn are demonstrated in details. A custom multiprocessor architecture can be generated by using Archborn APIs. Archborn templates provide a more convenient method and interface to build template architectures (e.g., RING, NUMA, and SMP). A targeted multiprocessor architecture can be built with either Archborn APIs or Archborn templates. The Archborn generation flow is based on bus hierarchies to expand the system. TCL codes shown in Figure 7 demonstrate the generation TCL scripts of Hthreads and HOpenCL platforms by using Archborn templates as the
interface. template_gen will return a bus list object where the host bus and multi-level group buses are recoded. Custom IPs and other vendor-provided IPs are added to corresponding buses by iterating the bus list.

Both Hthreads and HOpenCL hardware platforms are inherited from the NUMA architecture. TCL scripts shown as examples in Figure 7(a) and Figure 7(b) generate six groups in each of which both Hthreads and HOpenCL platform contain four processing elements. Since the first element of the returned bus list refers to the host bus, and remaining ones are group buses, therefore, IP modules can be connected to host and group buses respectively. For HOpenCL platform shown in Figure 7(b), four hardware kernels as PR modules are added to each group bus. In this example, we use the matrix multiplication as hardware accelerators.

C. Experimental Results

Our intent for Archborn is not to reveal the best architecture for any application but to provide a rapid automated generation tool. In this section, we verified the correctness of Archborn by using Hthreads and HOpenCL frameworks. We also presented the creation of CHMP systems with different programming models. Based on Archborn template, a large base CHMP system is built rapidly. Researcher can extend or modify the base CHMP system for specific needs.

Table II shows the resource utilization of both Hthreads and HOpenCL frameworks built with Archborn templates. In this experiment, we use Xilinx tool chain Vivado with VC707 board. We evaluate various numbers of PEs ranging from 1 to 64. Since the AXI interconnections can accommodate up to 16 slaves or 16 masters modules, in order to hold as many as PEs, we can either add multi-level group buses or connect more PEs to each group bus. Since HOpenCL will utilize group memories for fast data access within one group, there is no need to have only one or two PEs within one group.

We use the 2-dimensional integer matrix multiplication as the benchmark to evaluate the scalability of the multiprocessor architectures with supports of Hthreads and HOpenCL, respectively. The inputs of our benchmark are two 512 × 512 matrices. The output result is also a 512 × 512 matrix. We divided the output matrix into multiple groups that can be arranged
TABLE II. RESOURCE UTILIZATION OF HTHREADS AND HOOPENCL PLATFORMS WITH VARIOUS PES

<table>
<thead>
<tr>
<th># of PEs</th>
<th>Hthreads*</th>
<th>HOOpenCL†</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LUTs</td>
<td>BRAM</td>
</tr>
<tr>
<td>1</td>
<td>25,009 (8.0%)</td>
<td>36.0 (3.5%)</td>
</tr>
<tr>
<td>2</td>
<td>28,919 (9.5%)</td>
<td>45.5 (4.4%)</td>
</tr>
<tr>
<td>4</td>
<td>35,449 (11.7%)</td>
<td>65.5 (6.4%)</td>
</tr>
<tr>
<td>8</td>
<td>49,328 (16.3%)</td>
<td>105.5 (10.2%)</td>
</tr>
<tr>
<td>16</td>
<td>76,926 (25.3%)</td>
<td>185.5 (18.0%)</td>
</tr>
<tr>
<td>32</td>
<td>132,274 (43.6%)</td>
<td>345.5 (33.54%)</td>
</tr>
<tr>
<td>64</td>
<td>240,778 (79.3%)</td>
<td>695.5 (64.61%)</td>
</tr>
</tbody>
</table>

*The total number of PEs equals to the number of groups when it ranges from 1 to 8. When the total number of PEs is from 16 to 64, each group contains 8 PEs.
†Each group contains 4 PEs when the total number of PEs is from 4 to 32. 64 PEs require 8 groups, each of which contains 8 PEs.

to every processing element for computation. Depending on whether the actual hardware architecture has group memories (e.g., if one group contains multiple PEs, these PEs can share one group memory), partial input data can be fetched into group memories by DMAs for fast access. Figure 8 demonstrates the scalability of generated Hthreads and HOOpenCL platforms. The total number of PEs for each platform is from 4 to 24 with 4-stride. Both Hthreads and HOOpenCL platforms demonstrate the steady and similar scalability as the number of PEs increases.

V. CONCLUSION

Modern platform FPGAs are capable of accommodating a complete CHMP system including general-purpose processors, vendor-provided IPs and custom hardware accelerators. For system architecture researchers, it is necessary to provide a multiprocessor generation tool for rapid validation and system generation. Besides, a unified interface to embed modern programming models is essential to explore both DPL and TPL in CHMP.

In this work, we present Archborn, an open source multiprocessor architecture generation tool based on TCL scripts on platform FPGAs, to unify different hardware multiprocessor architectures with various programming models for rapid system generation. According to bus hierarchies within CHMP, we extract and wrap common operations for multiprocessor system generations as Archborn APIs. The Archborn templates simplify the generation flow to build common CHMP architectures. We take Hthreads and HOOpenCL as targeted programming models to present and verify the correctness of Archborn. The resource utilization and scalability of CHMP with up to 64 PEs are evaluated. A matrix multiplication benchmark is parallelized and executed on platform extended from the built CHMP by using two programming models. As an open source tool, Archborn will be released on public repository.

Although Archborn can significantly improve the productivity to generate a CHMP, there are two limitations. Firstly, users need to be familiar with vendor-specific TCL libraries. Secondly, current Archborn APIs follow function programming manners to manage IP resources. In the future work, Archborn will introduce the object-oriented programming (OOP) method to allow users to build specific architecture and write application source code using the same OOP language, such as C++ or Java.

ACKNOWLEDGMENT

This work is supported in part by NSF under grant CNS-1219062. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. The authors are grateful to Xilinx, Inc. for the donation of the FPGA boards and the software.

REFERENCES